NYSE +0.31% · NASDAQ +0.47% · LSE -0.12% · Eurex +0.18% · JPX +0.22% · HKEX -0.08% · CME +0.55% · SGX +0.11%



EQUITY TRADING INFRASTRUCTURE

QUICK START GUIDE

How to Get Started with EduMatcher

MARKET DEPTH · AAPL +1.24 +0.83%

BID QTY		PRICE	ASK QTY
8,340		150.35	3,920
7,210		150.34	5,100
5,820		150.33	6,400
12,100	LAST	150.32	7,840
4,420		150.31	10,220
3,200		150.30	12,800

EduMatcher Quick Start Guide
For EduMatcher Version: v0.77.0
Copyright © Johan Persson
Generated on November 8, 2026
Prepared with LaTeX from Markdown sources for duplex printing. Sources for documentation available at https://github.com/johan162/EduMatcher.

Quick Start Guide

`edumatcher`, Version: v0.77.0

From a first launch to a first trade

Johan Persson <johan162@gmail.com>

Generated on November 8, 2026

Chapters

1 How to Use This Book 1

2 What EduMatcher Is 5

3 Install and Start 9

4 Your First Trade 15

5 The Browser Applications 23

6 Run a Trading Session 27

7 Your Own Configuration 31

8 Choose Your Next Book 35

9 When Something Does Not Work 39

10 FAQ & Common Mistakes 43

11 Quick Glossary 49

Contents

1 How to Use This Book **1**
- 1.1 What you will be able to do . . . 1
- 1.2 How the book is organized . . . 1
- 1.3 The EduMatcher library . . . 2
- 1.4 Conventions . . . 3

2 What EduMatcher Is **5**
- 2.1 A working exchange, built for learning . . . 5
- 2.2 The system in one picture . . . 6
- 2.3 Five ideas to learn first . . . 6
- 2.4 Three details that surprise newcomers . . . 7
  - 2.4.1 An empty order book is normal . . . 7
  - 2.4.2 Prices are whole numbers of ticks inside . . . 7
  - 2.4.3 Trading dates are not calendar dates in UTC . . . 8
- 2.5 Where to go next . . . 8

3 Install and Start **9**
- 3.1 Before you start . . . 9
- 3.2 Pick a route . . . 9
- 3.3 Route 1: containers . . . 10
  - 3.3.1 Install and start . . . 10
  - 3.3.2 Checkpoint: open the browser applications . . . 10
  - 3.3.3 What you just started . . . 11
  - 3.3.4 A terminal inside the exchange . . . 11
- 3.4 Route 2: the Python package . . . 12
- 3.5 Where your data lives . . . 12
- 3.6 Where to go next . . . 13

4 Your First Trade **15**
- 4.1 Step 1 — Prepare an empty market . . . 15
- 4.2 Step 2 — Connect two traders . . . 16
- 4.3 Step 3 — See what you can trade . . . 16
- 4.4 Step 4 — Place a buy order that rests . . . 17
- 4.5 Step 5 — Sell into it . . . 17
- 4.6 Step 6 — Read the fill confirmation . . . 17
- 4.7 Step 7 — Check your position and P&L . . . 18
- 4.8 Step 8 — Close the position at a profit . . . 18
- 4.9 Step 9 — Cancel a resting order . . . 19

4.10 Step 10 — Amend a resting order . . . 19
4.11 Step 11 — Send a market order . . . 20
4.12 Summary . . . 20
4.13 Where to go next . . . 21

5 The Browser Applications 23
5.1 Six windows on one exchange . . . 23
5.2 Find your API key . . . 24
5.3 Trade from the browser . . . 24
5.4 When a display looks empty . . . 25
5.5 Where to go next . . . 25

6 Run a Trading Session 27
6.1 A trading day has phases . . . 27
6.2 Step 1 — Switch to a configuration with a timetable . . . 28
6.3 Step 2 — Open the operator console . . . 28
6.4 Step 3 — Pre-open: orders wait . . . 29
6.5 Step 4 — The opening auction and the uncross . . . 29
6.6 Step 5 — Look at the records . . . 30
6.7 Checkpoint . . . 30
6.8 Where to read more . . . 30

7 Your Own Configuration 31
7.1 The one idea to understand first . . . 31
7.2 Step 1 — Generate a starting point . . . 31
7.3 Step 2 — Check it . . . 32
7.4 Step 3 — Deploy and restart . . . 32
7.5 Checkpoint . . . 33
7.6 Where to read more . . . 33

8 Choose Your Next Book 35
8.1 The library at a glance . . . 35
8.2 Reading paths by role . . . 36
8.2.1 Student learning to trade . . . 36
8.2.2 Instructor running a class . . . 36
8.2.3 Market maker . . . 36
8.2.4 Operator or supervisor . . . 36
8.2.5 Analyst or auditor . . . 36
8.2.6 Developer of a client, feed handler or dashboard . . . 37
8.2.7 Contributor to EduMatcher itself . . . 37
8.3 Depth by interest . . . 37

9 When Something Does Not Work 39
9.1 Quick checks . . . 39
9.2 Common first-day problems . . . 39
9.3 Still stuck? . . . 41

10 FAQ & Common Mistakes 43
10.1 Architecture & Setup . . . 43
10.1.1 The engine started but gateways can't connect. What's wrong? . . . 43
10.1.2 The book already has liquidity before I submit any orders. Why? . . . 43

10.1.3 Why is the engine not matching orders? It starts in CLOSED state 43
10.1.4 Why is my viewer empty even though it started correctly? 44
10.1.5 Do I really need a terminal window for every process? 44
10.1.6 How do I reset all persisted state to start fresh? 44
10.2 Orders, Positions & Sessions 45
10.2.1 Why is my LIMIT order filling at a different price than I specified? 45
10.2.2 I submitted a MARKET order but it only partially filled. Where did the rest go? 45
10.2.3 Why is my unrealized P&L different from my realized P&L? 45
10.2.4 Why do I have a negative position? 45
10.2.5 What happens to my open orders when the market closes? 45
10.3 Installation 46
10.3.1 Do I need Poetry or pipx? 46
10.3.2 I installed with pipx but `pm-engine` is not found. What do I do? 46
10.3.3 What does `pm-setup` do and do I have to run it? 46
10.3.4 How do I upgrade EduMatcher to a newer version? 46
10.4 More questions 47

11 Quick Glossary **49**

CHAPTER 1

How to Use This Book

Welcome to EduMatcher. This Quick Start Guide takes you from nothing to a running exchange, a trade you made yourself and a clear idea of where to go next. Part I takes about an hour; Part II is another one to two hours, so plan two sittings.

Who it is for	Anyone new to EduMatcher: students, instructors trying the system, developers
What you need	A Linux or macOS computer (on Windows, a WSL 2 Linux terminal) with Podman or Docker, or with Python 3.14 and `pipx`; a web browser; and a terminal
What you do not need	Any finance background, Python knowledge or a copy of the source code
Time	About 15 minutes to a running exchange, about an hour to finish Part I, one to two more hours for Part II

1.1 What you will be able to do

After this book you can:

1. explain in a few sentences what EduMatcher is and how its parts fit together;
2. install and start the whole exchange, including its browser applications;
3. put an order into an order book, see it rest, match it, cancel and amend orders, and read your position and profit or loss;
4. run the exchange through a trading day with an opening and a closing phase;
5. write, check and deploy a configuration of your own;
6. pick the book that matches what you want to do next.

1.2 How the book is organized

Part	Chapters	What you get
I. See it run	What EduMatcher Is; Install and Start; Your First Trade; The Browser Applications	A running exchange and your first trades
II. Next steps	Run a Trading Session; Your Own Configuration; Choose Your Next Book	A full session, your own venue, a route onward
Appendices	When Something Does Not Work; FAQ; Quick Glossary	Help when you are stuck, and the words you need

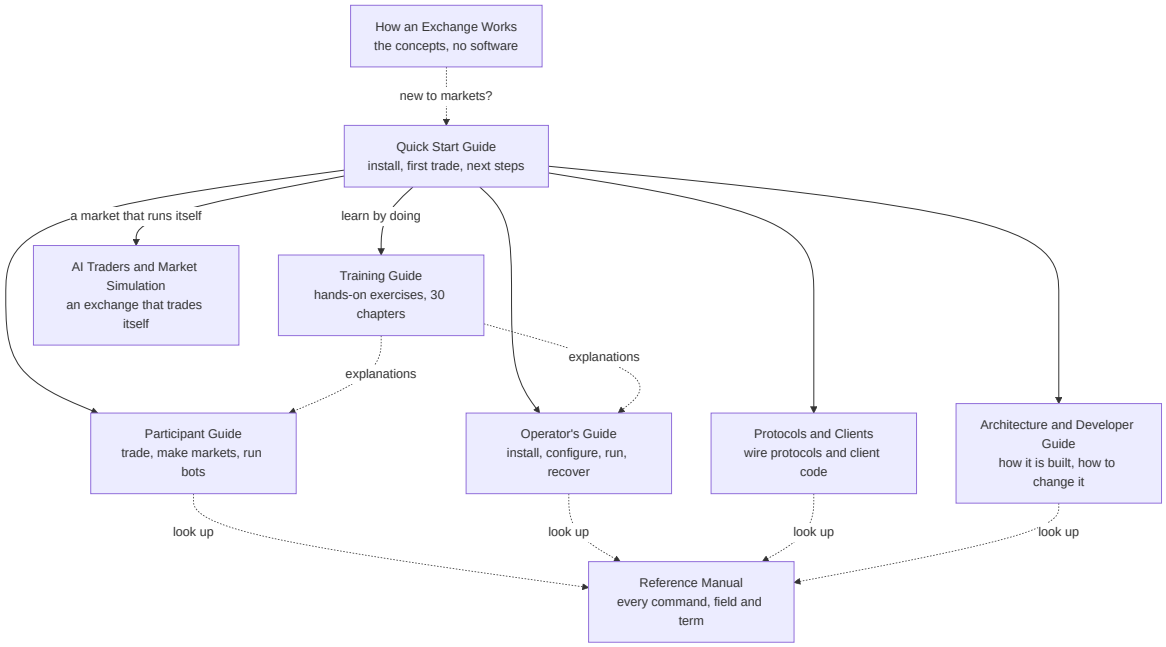
Each chapter starts with what it will teach you and ends with a checkpoint or a short list of where to read more. Nothing in a later chapter is needed to finish an earlier one.

New to exchanges altogether?

Read How an Exchange Works first, or alongside this book. It explains order books, bids and asks, auctions and market makers without assuming you know any of them, and makes everything here easier to follow.

1.3 The EduMatcher library

This guide is one of nine books. You do not have to read them all — most people read this one, then one or two others that match their role.



Book	Read it when you want to…	Written for
How an Exchange Works	understand exchanges in general, before touching any software	anyone new to markets
Quick Start Guide (this book)	get started	everyone, first

Book	Read it when you want to…	Written for
Training Guide	learn by doing, one exercise at a time	students, self-learners, classes
Participant Guide	trade, quote as a market maker, run trading bots, use the trading screens	traders and students
Operator's Guide	install, configure, run, supervise and recover an exchange	instructors and operators
Protocols and Clients	connect your own program to the exchange	client and tool developers
Architecture and Developer Guide	understand or change how EduMatcher is built	contributors
Reference Manual	look up one command, option, configuration field or term	everyone, mid-task
AI Traders and Market Simulation	run a simulated market — AI traders, hidden values, news — for a class, for days or weeks	instructors, curious students

Choose Your Next Book at the end of this guide gives a reading path for each role.

1.4 Conventions

- A **terminal** is the window where you type commands (Terminal on macOS, any terminal program on Linux, a WSL 2 window on Windows). A line starting with `$`, or a bare command in a `bash` block, is typed there and finished with Enter. Several chapters use two or three terminal windows at once.
- `[TRADER01]>` is the prompt of a **trader console** (`pm-alf-console`); type the text after it there.
- `[OPS01|ADMIN]>` is the prompt of the **operator console** (`pm-admin`).
- Output shown in a plain block is what you should see. Times, order IDs and some spacing will differ on your machine.
- `!!! tip`, `!!! note` and `!!! warning` boxes hold advice you can skip on a first reading; **Checkpoint** paragraphs tell you how to know a step worked.

CHAPTER 2

What EduMatcher Is

Learning objectives

After reading this chapter you will know:

- what EduMatcher is and what you can learn with it
- why it runs as many small `pm-*` programs instead of one
- the five ideas that make the rest of the documentation readable
- three details that surprise almost every newcomer

2.1 A working exchange, built for learning

EduMatcher is an educational stock exchange that really works. It has a matching engine, order books, a trading day with opening and closing auctions, market makers, risk controls, statistics, profit-and-loss tracking, an audit trail, external gateways and market-data feeds, monitoring tools and automated trading bots. Its architecture and behavior stay close to those of a real exchange.

That makes it useful for three kinds of learning:

- **Market microstructure** — what happens inside an order book, why auctions exist, and how spreads, time priority, market makers and risk controls shape a market.
- **Exchange operations** — how to configure a trading venue, start it, run a trading day, watch it and find out afterwards what happened.
- **Protocol and system design** — how order entry, market data, post-trade feeds, logging and recovery are separated in a system of cooperating processes.

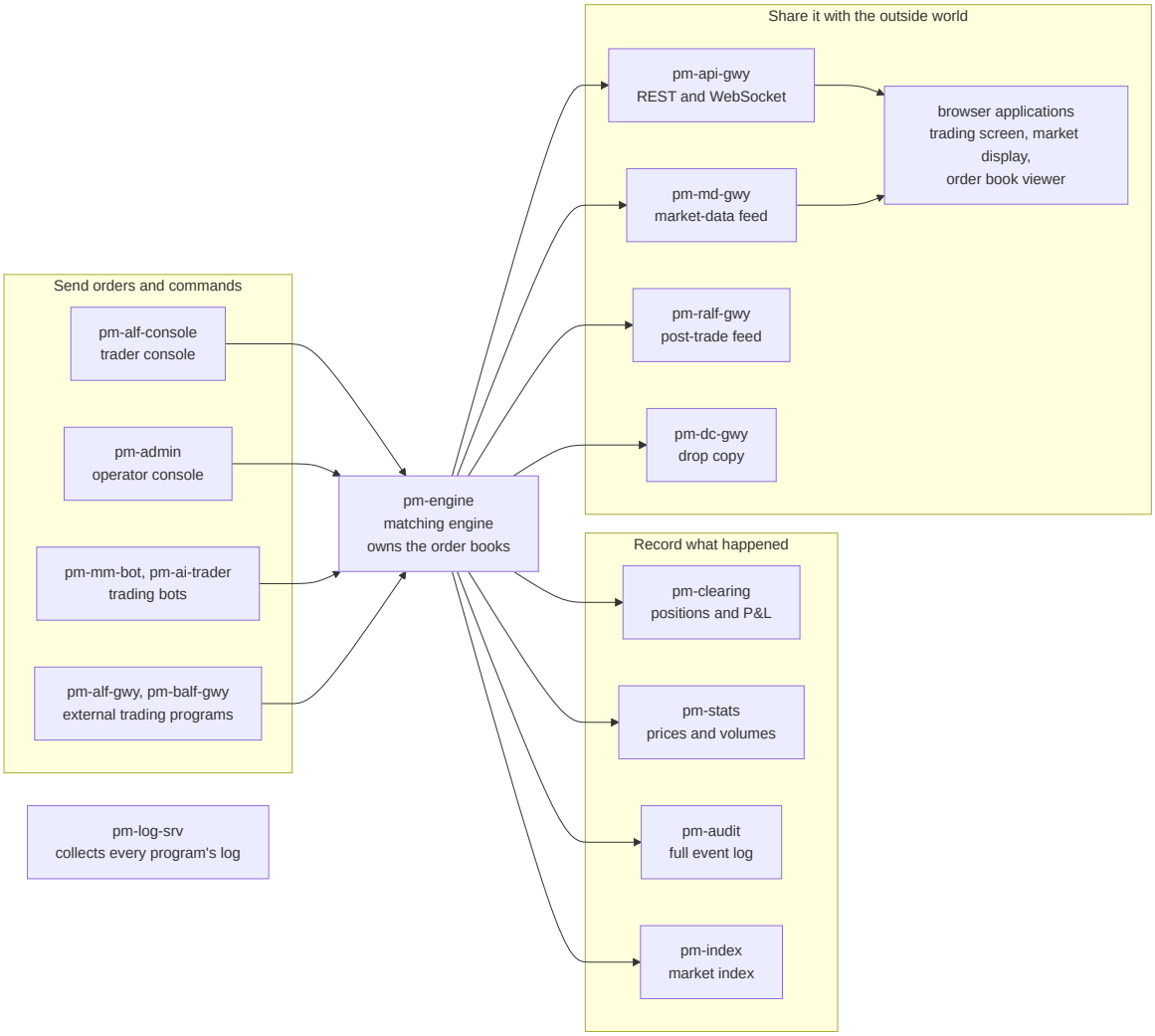
Some things are deliberately simpler than on a real exchange so that they do not get in the way of learning: authentication is minimal, there is no member and user hierarchy, and there is no standby system or other fault tolerance. The Reference Manual's Known Limitations lists the rest. EduMatcher is for teaching and experimenting — **never use it for real money**.

The documentation is large; your first steps are not

EduMatcher covers a whole exchange, so its documentation is long. You do not need most of it to begin. One process, one symbol and two orders are enough for a first trade, and that is where this guide takes you next.

2.2 The system in one picture

EduMatcher is a set of independent programs, all named `pm-*`, that talk to each other over a message bus. Only one of them, the **engine**, owns the order books and decides what trades. Every other program sends instructions to the engine, listens to what it announces, or passes that information on to someone else.



The first idea to take away is that **the exchange is not one program**. It is a small operating environment. A five-minute demonstration needs only the engine and two trader consoles. A classroom session adds a trading-day schedule, recorders for prices and profits, market data, logging and the browser displays. The container installation in the next chapter starts all of them for you with one command.

Why split it up this way? Real exchanges do, for the same reasons: each part can fail, restart or be replaced without stopping the others, and the engine — the part that must be correct and fast — stays small. The Architecture and Developer Guide tells that story in Architecture.

2.3 Five ideas to learn first

These five ideas are enough to make the rest of the documentation readable. The first three appear in the next two chapters; session phases come in Run a Trading Session and the deployed configuration in Your Own Configuration.

Idea	What it means
Engine	`pm-engine`, the one program that owns all order books and matches orders. If the engine is not running, nothing trades.
Symbol	A tradeable instrument, such as `AAPL`. Each symbol has its own order book, its own price step (the *tick size*) and its own risk settings.
Participant	Anyone who connects to trade or to operate: a student, a bot, the instructor. Each has an ID such as `TRADER01` and a **role** — `TRADER`, `MARKET_MAKER` or `ADMIN` — that decides what it may do. On the wire and in the consoles, the participant ID is also called the **gateway ID**.
Session phase	Where the trading day is: `PRE_OPEN`, `OPENING_AUCTION`, `CONTINUOUS`, `CLOSING_AUCTION` or `CLOSED` (plus phases for halts). Orders are treated differently in each.
Deployed configuration	One configuration file describes the whole exchange: its symbols, participants, trading day and rules. You edit it as YAML and *deploy* it; every program then reads the same compiled copy.

Two more ideas matter once you start looking at results:

- **Events are not records.** The engine *announces* what happens as it happens. Recorder programs such as `pm-stats`, `pm-clearing` and `pm-audit` turn those announcements into files and databases — but only while they are running. Anything that happens while a recorder is stopped is not in its records.
- **Inside and outside are separate.** The `pm-*` programs talk to each other over an internal message bus. Programs written by other people connect from outside through **gateways** that speak documented protocols (ALF, BALF, CALF, RALF, drop copy, REST and WebSocket). The Protocols and Clients book describes them.

2.4 Three details that surprise newcomers

2.4.1 An empty order book is normal

An exchange never invents prices. A book has buyers and sellers only when someone has put orders into it. If you send an order and nothing happens, the most likely reason is simply that nobody is waiting on the other side at a price that matches. Liquidity comes from other participants, from **market makers** whose job is to quote both a buying and a selling price, or from the bots EduMatcher provides.

2.4.2 Prices are whole numbers of ticks inside

You see prices like `150.25`. Inside, the engine stores and matches whole numbers of *ticks*: with two decimals, `150.25` is `15025` ticks. This avoids rounding errors in matching and in totals. The commands, screens and APIs convert for you; only raw database rows show the tick form.

2.4.3 Trading dates are not calendar dates in UTC

Every event carries a precise UTC timestamp, but daily statistics and profit summaries are grouped by the exchange's own **trading date** in its local time zone. A session that runs across midnight UTC can therefore have one trading date and two UTC dates. This only matters when you compare daily reports, and the Sessions and Scheduling chapter explains it in full.

2.5 Where to go next

Continue with Install and Start. If any of the words above were new, keep the Quick Glossary open alongside.

CHAPTER 3

Install and Start

Learning objectives

After this chapter you will have:

- a running exchange, started with one command
- the browser applications open and showing live quotes
- a terminal "inside" the exchange where every `pm-*` command works
- an idea of where EduMatcher keeps its data, and how to stop and start it

Time: about 15 minutes, most of it spent downloading.

3.1 Before you start

You need a Linux or macOS computer. On Windows, install WSL 2 and run every command in this book in its Linux terminal: the installer and the `edumatcher.sh` control script are bash scripts. You also need a web browser and an internet connection for the first download of the container images.

Check what you already have by typing these in a terminal:

```
podman --version          # either of these two is enough ...
docker --version
docker compose version    # ... but Docker also needs Compose
python3 --version         # only for the Python route; it must say 3.14 or newer
```

A "command not found" simply means that tool is not installed. If you have neither Podman nor Docker, install one first (on macOS, Docker Desktop or Podman with `podman machine start`; on Linux, your package manager). Most people should then use the container route below.

3.2 Pick a route

There are two ways in. Both give you the same exchange.

	Containers (recommended)	**Python package** (`pipx`)
You need	Podman or Docker, with Compose	Python 3.14 and `pipx`
Time to a running exchange	one command	a few commands
Browser applications	six, ready to use	not included

	Containers (recommended)	**Python package** (`pipx`)
Where `pm-*` commands run	inside the container, after `./edumatcher.sh shell`	in your own terminal
Your data lives in	`~/.edumatcher/data`	`~/.local/share/edumatcher`
Best for	seeing the whole system; classes and demos	learning the processes one by one

If you are unsure, use containers: nothing is installed outside one directory you can delete again, and you get the browser applications for free. The Operator's Guide Installation chapter describes these and three more routes (a Multipass VM, building the containers from source, and a developer checkout) in full detail.

3.3 Route 1: containers

3.3.1 Install and start

With Podman or Docker installed, run:

```
curl -fsSL https://raw.githubusercontent.com/johan162/EduMatcher/main/deployment/curl/install.sh | bash
cd ~/.edumatcher
```

(The installer is a short script that you are piping into `bash`. If you prefer to read it first, open the URL in a browser, or download it with `curl -fsSLO <the same URL>`, look through it and run `bash install.sh`.)

The installer downloads the latest release into `~/.edumatcher`, pulls seven container images (the exchange plus six browser applications) and starts them. The first start takes a few minutes; later starts take seconds. When it finishes it prints the addresses of the browser applications.

`./edumatcher.sh` is your control panel for this installation. You only need a handful of its commands:

Command	What it does
`./edumatcher.sh status`	Shows the containers and every exchange process with its health
`./edumatcher.sh urls`	Prints the browser addresses again
`./edumatcher.sh shell`	Opens a terminal inside the exchange, where the `pm-*` commands live
`./edumatcher.sh logs`	Follows what the containers print, starting with the startup messages; press `Ctrl-C` to stop watching (the exchange keeps running)
`./edumatcher.sh config <name>`	Chooses another configuration (applied by the next `restart`)
`./edumatcher.sh restart`	Stops and starts everything, applying configuration changes
`./edumatcher.sh stop` / `start`	Stops or starts everything; your data is kept

3.3.2 Checkpoint: open the browser applications

Open these addresses on the same computer:

Address	Application	What you should see
http://localhost:8090	**TapeDeck** — the market display	Ten symbols (`AAPL`, `MSFT`, `TSLA`, …) with bid and ask prices
http://localhost:8094	**Order Book Viewer**	One symbol's full book; press `s` to switch symbol
http://localhost:8093	**Trading GUI**	A login screen — you will use it in The Browser Applications
http://localhost:8091	**Log Operator Console**	The operational log of every process
http://localhost:8095	**Operator Dashboard**	Every process HEALTHY and the exchange OPERATIONAL
http://localhost:8092	**Configuration GUI**	A form-based editor for exchange configurations
http://localhost:8080/docs	**REST API documentation**	The interactive Swagger page of the API gateway

Then confirm that the exchange itself is healthy:

```
./edumatcher.sh status
```

Every process in the table should show `running`. If a container is missing or a page does not open, see When Something Does Not Work.

Where do the prices on TapeDeck come from?

The container starts a slow **market-maker bot**, `pm-mm-bot`, trading as participant `MM01`. A market maker's job is to always offer to buy (a *bid*) and to sell (an *ask*), so the books are never empty. Nobody is trading yet, so you see quotes moving now and then, but no trades. You will make the first trade in the next chapter.

3.3.3 What you just started

One container runs the exchange: the engine and about fifteen supporting processes (sixteen in all, including the market-maker bot), started in the right order by the process manager `pm-opctl-cli`. Six more containers run the browser applications. They share a private network, which is why nothing asked you for an address. The bundled configuration is called `s10-basic`: ten symbols, four participants (`TRADER01`, `TRADER02`, `MM01` and the operator `OPS01`) and no trading-day schedule, so the market is open for continuous trading straight away.

3.3.4 A terminal inside the exchange

The `pm-*` commands are installed inside the exchange container, not on your computer. Open a shell there:

```
./edumatcher.sh shell
```

Your prompt changes, and every `pm-*` command now works. Try the built-in command index:

```
pm-help                 # one line for every pm-* command, grouped by purpose
pm-help pm-alf-console  # the full manual page of one command
pm-config-show          # what the running exchange is configured with
```

Leave the shell with `exit` or `Ctrl-D`. You can open as many as you like, one per terminal window — the next chapter uses two.

3.4 Route 2: the Python package

Use this route if you would rather run each process yourself, or cannot use containers. It needs **Python 3.14 or newer**, which many systems do not have by default (check with `python3 --version`); if yours is older, use the container route instead.

`pipx` installs Python programs in their own environments. If `pipx --version` says "command not found", install it with your package manager (for example `brew install pipx` on macOS, or `python3 -m pip install --user pipx`) and then run `pipx ensurepath`. Open a new terminal window afterwards so the change takes effect. Then:

```
pipx install edumatcher
pm-setup
```

If you have several Python versions, tell pipx which to use: `pipx install --python python3.14 edumatcher`.

`pm-setup` creates the data directory (`~/.local/share/edumatcher`) and deploys the bundled `s10-basic` configuration. It also prints one line to add to your shell profile:

```
export EDUMATCHER_DATA_DIR="$HOME/.local/share/edumatcher"
```

Every `pm-*` process finds the configuration, and writes its databases and logs, through this one variable, so all processes of one exchange must agree on it.

Start the whole exchange with the process manager, and check it:

```
pm-opctl-cli start      # starts the full set of processes in the right order
pm-opctl-cli list       # one line per process, with its health
```

`pm-opctl-cli stop` stops them all again. The next chapter also shows how to start just the two or three processes a first trade needs, one terminal each — the slow way, and the best way to see what every process does.

Browser applications on the Python route

The browser applications are web projects of their own and are not part of the Python package, but you can run them as ready-built containers next to your `pipx` exchange (a plain exchange or the simulation). With Podman or Docker installed, and the exchange started with `EDUMATCHER_GATEWAY_BIND_HOST=0.0.0.0 pm-opctl-cli start`:

```
curl -fsSL https://raw.githubusercontent.com/johan162/EduMatcher/main/deployment/curl/install.sh \
| bash -s -- --guis
```

The addresses are the same as on the container route. The Operator's Guide chapter Installation explains what it sets up and what to check if a GUI stays empty.

3.5 Where your data lives

Everything the exchange produces — order books saved between sessions, statistics, profit-and-loss records, the audit trail, logs — is an ordinary file in one data directory:

Route	Data directory
Containers	`~/.edumatcher/data` (seen as `/data` inside the container)
Python package	`~/.local/share/edumatcher`, or `$EDUMATCHER_DATA_DIR` if set

The deployed configuration is inside it, at `ref_data/engine_config.json`.

Why the two routes use different directories

The Python code picks the directory the same way on both routes; the container simply sets `EDUMATCHER_DATA_DIR=/data` and mounts the `data` folder of your install directory there. The routes keep their data apart on purpose:

- **One folder per install.** `~/.edumatcher` holds the script, its settings and the data, so deleting that one folder removes the whole installation, and installs made with `--dir` never share data.
- **No clashes.** A container and a `pipx` install sharing one directory would overwrite each other's deployed configuration and databases, and different versions may not accept each other's compiled configuration.
- **File ownership.** With Docker on Linux the container writes its files as `root`, which your own `pm-*` commands could then not change.

The consequence: a `pm-*` command you installed with `pipx` does **not** see the container's exchange data by default. Use `./edumatcher.sh shell` instead, or point the command at the container's data explicitly:

```
export EDUMATCHER_DATA_DIR="$HOME/.edumatcher/data"
```

The Reference Manual's Environment variables section explains exactly how the directory is chosen, and the Operator's Guide chapter Persistence lists every file in it.

3.6 Where to go next

Continue with Your First Trade. If something did not work, see When Something Does Not Work.

CHAPTER 4

Your First Trade

Learning objectives

After this walkthrough you will have:

- put a limit order into an empty order book and watched it rest there
- made it trade with an order from a second participant
- read the fill, and worked out who was the *maker* and who the *taker*
- checked your position and your realized profit or loss
- cancelled and amended a resting order
- sent a market order

Time: about 30 minutes. **You need:** the exchange from Install and Start and three terminal windows. No trading knowledge is assumed — every term is explained where it first appears.

Two ideas before you start

An **order book** is the list of everyone currently willing to buy a symbol (the *bids*) and everyone willing to sell it (the *asks*), best price first. An order **rests** in the book until an order from the other side arrives at a price that crosses it — then the two **match** and a trade happens. If you would like a fuller picture first, read The Order Book in the Participant Guide; it takes ten minutes.

4.1 Step 1 — Prepare an empty market

To see your own orders clearly, start from a market with no other orders in it: no seeded quotes and no market-maker bot. The bundled configuration `s3-basic-nomm` has three symbols (`AAPL`, `MSFT`, `TSLA`), the same four participants as before, no trading-day schedule and *no market-maker quotes* ("nomm").

Containers. In `~/.edumatcher`, choose the configuration, switch the process profile from `mm-demo` to `default` (the same processes without the market-maker bot) and restart:

```
cd ~/.edumatcher
./edumatcher.sh config s3-basic-nomm
sed -i.bak 's/^EM_PROFILE=.*/EM_PROFILE=default/' .env
./edumatcher.sh restart
```

The `sed` line just edits one setting in the file `.env`; you can make the same change in any text editor. (A file whose name starts with a dot is hidden by default; `ls -a` shows it.) Check the result with `grep EM_PROFILE .env`, which should print `EM_PROFILE=default`. Then open **two** terminals and run `./edumatcher.sh shell` in each, so both are inside the exchange.

Python package. Deploy the same configuration and restart the processes:

```
pm-opctl-cli stop
pm-setup --config s3-basic-nomm --force
pm-opctl-cli start
```

Then open two terminals with the same `EDUMATCHER_DATA_DIR`.

The minimal exchange, by hand

On the Python route you can also skip `pm-opctl-cli` and start only what a trade needs: run `pm-engine --verbose` in a third terminal and leave it running. That is the whole matching engine. Everything below works the same, except Step 7's `pm-clearing-cli`, which needs the `pm-clearing` recorder running *before* you trade.

4.2 Step 2 — Connect two traders

In the first terminal, connect as `TRADER01`; in the second, as `TRADER02`:

```
pm-alf-console --id TRADER01
```

```
pm-alf-console --id TRADER02
```

`pm-alf-console` is the **trader console**: it connects you to the engine as one participant, using the text order-entry protocol ALF. Each window prints a banner and a prompt:

```
Gateway TRADER01 connected.   — Student desk 1 Type HELP for commands.
Tab=complete  ↑↓=history  Ctrl-A/E=line start/end

[TRADER01]>
```

Everything the engine sends back to you — acknowledgements, fills, rejections — appears in the same window with a `[HH:MM:SS.mmm]` timestamp. Type `HELP` at any time to list the commands.

Leaving a console cancels your orders

In these configurations a trader's resting orders are cancelled when the trader disconnects. If you type `EXIT` or close the window, everything TRADER01 had in the book disappears, and a later step that expects it will not find it. Keep both consoles open until you finish the chapter. A participant ID can also be connected only once at a time: starting a second console as `TRADER01` (or logging in to the Trading GUI with its key) is refused with *Gateway already connected.*

4.3 Step 3 — See what you can trade

At the `[TRADER01]>` prompt, type:

```
SYMBOLS
```

```
                        Active Instruments
┌─────┬──────────────┬──────┬──────────────┬────────────┬─────────┐
│ #   │ Symbol       │ Tick │ MM Enforced  │ Max Spread │ Min Qty │
├─────┼──────────────┼──────┼──────────────┼────────────┼─────────┤
│ 1   │ AAPL         │ 0.01 │      NO      │         20 │     100 │
│ 2   │ MSFT         │ 0.01 │      NO      │         20 │     100 │
│ 3   │ TSLA         │ 0.01 │      NO      │         20 │     100 │
└─────┴──────────────┴──────┴──────────────┴────────────┴─────────┘
```

`Tick` is the smallest price step: AAPL prices are whole cents. The other columns are market-maker rules, which this configuration does not enforce; in particular `Min Qty` is a market-maker quoting rule,

not a minimum for your orders, so the 50- and 30-share orders in later steps are fine. Every example below uses `AAPL`.

4.4 Step 4 — Place a buy order that rests

A **limit order** says: *"I want to buy (or sell) this many shares, but only at this price or better."* At the `[TRADER01]>` prompt:

```
NEW|SYM=AAPL|SIDE=BUY|TYPE=LIMIT|QTY=100|PRICE=150.00|TIF=DAY
```

Every command is a word (`NEW`) followed by `KEY=VALUE` pairs, all separated by the `|` character, with no spaces. Type it exactly like this, in capital letters; the Tab key completes command names and ↑ brings back the previous line. Prices are written with a decimal point.

Field	Value	Meaning
`SYM`	`AAPL`	The symbol to trade
`SIDE`	`BUY`	You want to buy
`TYPE`	`LIMIT`	Never pay more than `PRICE`
`QTY`	`100`	100 shares
`PRICE`	`150.00`	The most you will pay per share
`TIF`	`DAY`	*Time in force*: the order expires at the end of the trading day if it has not traded

The engine acknowledges it:

```
[09:31:02.104] ACK      855b6f946fe9cb7ae96bbda1c8d1dc06  order accepted
```

The long hexadecimal string is the **order ID**. You will need it to cancel or amend an order, so it is worth knowing where to find it: here, and in the `ORDERS` list you will use in Step 9.

Nobody is selling, so nothing trades. Your order now **rests** in the book as the best bid. If you have the container route, look at AAPL in the Order Book Viewer (http://localhost:8094) or on TapeDeck (http://localhost:8090): a bid of 100 at 150.00 has appeared where there was nothing.

4.5 Step 5 — Sell into it

Switch to the `[TRADER02]>` window and offer to sell at the same price:

```
NEW|SYM=AAPL|SIDE=SELL|TYPE=LIMIT|QTY=100|PRICE=150.00|TIF=DAY
```

TRADER01 is willing to pay 150.00 and TRADER02 is willing to accept 150.00: the prices **cross**, so the engine matches the two orders at once instead of letting the sell order rest.

4.6 Step 6 — Read the fill confirmation

Both windows receive a `FILL` line the moment the trade happens. In `TRADER01`'s window:

```
[09:31:07.552] FILL     855b6f946fe9cb7ae96bbda1c8d1dc06  qty=100 @150.0  remaining=0  [FILLED]
```

TRADER02 sees the same for its own order ID. `remaining=0` and `[FILLED]` mean the whole order traded. A partial fill would show `remaining=<n>` and `[PARTIAL]`, and the rest of the order would keep resting in the book.

This is the moment to meet two words used everywhere on exchanges:

- TRADER01's order was already resting in the book and *provided* the liquidity: TRADER01 was the **maker**.
- TRADER02's order arrived and *took* that liquidity: TRADER02 was the **taker**.

You can ask the engine to state this for every fill. Start a console with `--drop-copy` (or type `DC|STATE=ON` in a connected one) and each of your fills is followed by a **drop-copy** line carrying the flag:

```
[09:31:07.552] DC_FILL   855b6f946fe9cb7ae96bbda1c8d1dc06  AAPL  qty=100 @150.0  [MAKER]  #1  (drop_copy.event.TRADE
```

Drop copy is the independent copy of fills that a firm's risk and compliance staff receive. The Participant Guide covers it in The Trader Console.

4.7 Step 7 — Check your position and P&L

Your **position** is how many shares you hold: positive after buying (*long*), negative after selling shares you did not have (*short*). At the `[TRADER01]>` prompt:

```
POS
```

```
                                Positions
┌──────────┬──────────┬──────────┬──────────┬──────────┬──────────┐
│ Symbol   │ Net Qty  │ Avg Cost │ Last Px  │ Unreal P&L │ Real P&L │
├──────────┼──────────┼──────────┼──────────┼──────────┼──────────┤
│ AAPL     │     +100 │   150.00 │   150.00 │    +0.00 │    +0.00 │
└──────────┴──────────┴──────────┴──────────┴──────────┴──────────┘
```

TRADER01 is long 100 shares bought at an average cost of 150.00. The last trade price is also 150.00, so the **unrealized** P&L (the gain or loss you would make by closing the position at the last price) is zero. TRADER02 is short 100, also at zero. Nothing is **realized** yet, because neither has closed a position.

4.8 Step 8 — Close the position at a profit

TRADER01 now offers the shares back at a higher price. From `[TRADER01]>`:

```
NEW|SYM=AAPL|SIDE=SELL|TYPE=LIMIT|QTY=100|PRICE=152.00|TIF=GTC
```

`TIF=GTC` means *good till cancelled*: unlike a `DAY` order, it is saved when the trading day ends and comes back the next day if it has not traded. It rests on the ask side of the book. From `[TRADER02]>`, buy back the short position at that price:

```
NEW|SYM=AAPL|SIDE=BUY|TYPE=LIMIT|QTY=100|PRICE=152.00|TIF=DAY
```

The orders cross and trade. Type `POS` again at `[TRADER01]>`:

```
| Symbol   | Net Qty  | Avg Cost | Last Px  | Unreal P&L | Real P&L |
| AAPL     |        0 |        – |        – |          – |  +200.00 |
```

TRADER01 bought at 150 and sold at 152: 2.00 × 100 shares = **200.00 realized profit**, and a flat position. TRADER02 sold at 150 and bought back at 152 and has realized a 200.00 loss. Every profit on an exchange is someone else's loss or forgone gain: the exchange itself only matches.

The same figures are recorded by the `pm-clearing` recorder, which the full process set starts for you. From any shell inside the exchange:

```
pm-clearing-cli pnl
```

```
gateway_id | symbol | realized_pnl | unrealized_pnl | total_pnl | net_qty | mark_price | tick_decimals
-----------+--------+--------------+----------------+-----------+---------+------------+--------------
TRADER01   | AAPL   | 200          | 0              | 200       | 0       | 152        | 2
TRADER02   | AAPL   | -200         | 0              | -200      | 0       | 152        | 2
```

Recorders only record what they see

`pm-clearing`, `pm-stats` and `pm-audit` only capture trades that happen while they are running. If you started the engine by hand and `pm-clearing` afterwards, its figures miss the earlier trades and will not match `POS`. That is why the process manager starts the recorders before the engine.

4.9 Step 9 — Cancel a resting order

Post a bid that will not trade straight away. From `[TRADER01]>`:

```
NEW|SYM=AAPL|SIDE=BUY|TYPE=LIMIT|QTY=50|PRICE=148.00|TIF=DAY
```

To cancel an order you need its ID. Copy it from the `ACK` line, or list your orders:

```
ORDERS
```

```
| ID                               | Symbol | Side | Type  | TIF | Qty | Rem | Price | Status | Time     |
| a129aadacb20eaf85c4eef86386be6e9 | AAPL   | BUY  | LIMIT | DAY |  50 |  50 | 148.0 | NEW    | 09:34:15 |
| 9ceca6d5db612b51d2dcb94ea880055f | AAPL   | SELL | LIMIT | GTC | 100 |   0 | 152.0 | FILLED | 09:32:40 |
| 855b6f946fe9cb7ae96bbda1c8d1dc06 | AAPL   | BUY  | LIMIT | DAY | 100 |   0 | 150.0 | FILLED | 09:31:02 |
```

Then cancel it with the full ID:

```
CANCEL|ID=a129aadacb20eaf85c4eef86386be6e9
```

```
[09:34:22.017] CANCELLED a129aadacb20eaf85c4eef86386be6e9
```

Use the whole ID

`CANCEL` and `AMEND` need the complete 32-character order ID. In a narrow terminal window the `ORDERS` table shortens the ID column (`a129aa…`); widen the window, or copy the ID from the `ACK` line instead.

4.10 Step 10 — Amend a resting order

Sometimes you want to change an order rather than withdraw it. Post a new bid, note its ID, and then change its price to 149.00 and its quantity to 30:

```
NEW|SYM=AAPL|SIDE=BUY|TYPE=LIMIT|QTY=50|PRICE=148.00|TIF=DAY
AMEND|ID=<the new order's ID>|PRICE=149.00|QTY=30
```

```
[09:35:40.229] AMENDED   73cfaf47645825b341c93513a908bd81  price=149.0 qty=30 remaining=30 (priority reset)
```

`(priority reset)` matters. Orders at the same price trade in the order they arrived — **price-time priority** — and an amendment can cost you your place in that queue, just as on real exchanges:

- **a price change** always sends the order to the back of the queue at its new price;
- **a quantity increase** does too;
- **a quantity decrease** keeps your place, because offering less is no disadvantage to anyone already waiting.

Here the price changed, so the order lost its priority.

4.11 Step 11 — Send a market order

A **market order** has no price: it says *"buy (or sell) now, at the best prices available"*. First give it something to trade against. From `[TRADER02]>`:

```
NEW|SYM=AAPL|SIDE=SELL|TYPE=LIMIT|QTY=100|PRICE=151.00|TIF=DAY
```

Then, from `[TRADER01]>`:

```
NEW|SYM=AAPL|SIDE=BUY|TYPE=MARKET|QTY=100
```

```
[09:36:51.703] FILL      53d7d9eae488049000d4131a1729cd07  qty=100 @151.0  remaining=0  [FILLED]
```

It filled at 151.00 — the resting seller's price, not one you chose. A market order trades certainty of execution for control over the price. In a thin book that can be an expensive trade, which is why the Participant Guide's Order Types chapter spends some time on it.

4.12 Summary

Step	What you did	What it taught you
1–2	Prepared an empty market and connected two traders	Participants, the trader console
3	Listed the symbols	Instruments and tick size
4	Posted a limit buy that rested	Bids, resting orders, order IDs
5–6	Sold into it and read the fills	Crossing prices, fills, maker and taker
7	Looked at your position	Long, short, unrealized P&L
8	Closed the position	Realized P&L, `GTC`, recorders
9	Cancelled an order	Order IDs, `ORDERS`
10	Amended an order	Price-time priority
11	Sent a market order	Immediacy versus price

Back to the full demo market

To get the market-maker bot and the ten-symbol configuration back on the container route, run `./edumatcher.sh config s10-basic`, set `EM_PROFILE=mm-demo` again in `.env`, and `./edumatcher.sh restart`.

Start over from a clean market

If your orders, positions or P&L no longer match this chapter (for example after a mistyped command), reset the exchange's saved state. On the container route:

```
./edumatcher.sh shell pm-opctl-cli stop
./edumatcher.sh shell pm-opctl-cli clear --state --yes
./edumatcher.sh restart
```

On the Python route, run the same `pm-opctl-cli stop`, `clear --state --yes` and then `pm-opctl-cli start`. The deployed configuration is kept.

4.13 Where to go next

- The Browser Applications — the same market through the trading screen and the market displays.
- Order Types — every order type, with its exact behavior.
- The Trader Console — every command you can type at the `[TRADER01]>` prompt.
- The Training Guide's The First Trade — the same ground as exercises with checkpoints.

CHAPTER 5

The Browser Applications

Learning objectives

After this chapter you will know:

- what each of the six browser applications is for
- how to log in to the Trading GUI with an API key, and where to find one
- how to place the same kind of order from the browser that you typed in the trader console

Time: about 15 minutes. **You need:** the container installation from Install and Start. The browser applications are not part of the Python package.

5.1 Six windows on one exchange

All six applications talk to the same running exchange, so anything you do in one — or in a `pm-alf-console` — shows up in the others within a second.

Address	Application	Read-only?	Use it to…
http://localhost:8090	**TapeDeck** (`pm-terminal`)	yes	watch prices, trades, auctions and halts across all symbols — a classroom wallboard
http://localhost:8094	**Order Book Viewer** (`pm-book`)	yes	see every price level of one symbol's book, its statistics and its trade tape
http://localhost:8093	**Trading GUI** (`pm-trading-ui`)	no	trade, quote or operate the exchange through screens instead of commands
http://localhost:8091	**Log Operator Console** (`pm-log-ui`)	yes	read what every process is doing, and why something was rejected

Address	Application	Read-only?	Use it to…
http://localhost:8092	**Configuration GUI** (`config-gui`)	—	build an exchange configuration with forms instead of YAML
http://localhost:8095	**Operator Dashboard** (`pm-dash`)	yes	see at a glance whether every part of the exchange is healthy, and how busy it is

The read-only displays need no login. The Trading GUI does, because it can change things.

What you will see depends on the previous chapter

If you followed Your First Trade, the exchange now runs `s3-basic-nomm`: three symbols and empty books, so TapeDeck shows `AAPL`, `MSFT` and `TSLA` without prices until someone trades. After `./edumatcher.sh config s10-basic` (see the tip at the end of that chapter) you get the ten-symbol market with quotes again.

5.2 Find your API key

The Trading GUI logs in with an **API key**. Each key in the configuration belongs to one participant, and the participant's role (`TRADER`, `MARKET_MAKER` or `ADMIN`) decides which screens you get. The bundled configurations generate their keys, so look them up in the running exchange:

```
./edumatcher.sh shell pm-config-show -a
```

The `-a` option shows everything, including the keys in full. Look for the **API KEYS** panel:

```
┌─ API KEYS ──────────────────────────────────────────────────────────┐
| GATEWAY ID      API GW        ROLE           API KEY                |
| TRADER01        desk          TRADER         key-trader01-z8hu5h    |
| TRADER02        desk          TRADER         key-trader02-199pf4    |
| OPS01           desk          ADMIN          key-ops01-tvj553       |
| MM01            desk          MARKET_MAKER   key-mm01-tp6aoi        |
| –               dashboards    READ-ONLY      key-readonly-z6f1hx    |
└─────────────────────────────────────────────────────────────────────┘
```

Your keys will be different, and each configuration has its own: after you switch configuration (as in the previous chapter, or in Your Own Configuration) look the keys up again, because the old ones stop working. The `READ-ONLY` key is the one the read-only displays (such as TapeDeck and the Order Book Viewer) use behind the scenes; it cannot log in to the Trading GUI.

5.3 Trade from the browser

1. Close any `pm-alf-console` that is connected as `TRADER01`: a participant ID can be connected only once at a time, and the Trading GUI counts. Then open http://localhost:8093, paste `TRADER01`'s key and click **Connect**. If the key is wrong or from an older configuration, you stay on the login screen; look it up again as above. You land in the **Trading Workspace**: a price chart, a depth ladder (the order book, best prices in the middle), an order ticket and a blotter of your working orders.
2. Open the Order Book Viewer at http://localhost:8094 in a second window, on the same symbol.

3. In the ticket, choose **Limit**, enter a quantity and a price below the best ask, and press **Buy**. A toast confirms *accepted*, the order appears in the blotter, and the Order Book Viewer shows it as a new bid.
4. Sell into it from a trader console connected as `TRADER02` (`pm-alf-console --id TRADER02` in `./edumatcher.sh shell`), or log in as `TRADER02` in a private browser window, entering the same quantity at the same price, and watch the trade appear on the tape in both windows and on TapeDeck.

Reloading logs you out

The Trading GUI keeps your key in memory only, never on disk. Reloading the page returns you to the login screen. That is deliberate, not a fault.
Log in with `OPS01`'s key instead and you get the operator's screens: the system dashboard, session control, halts, circuit breakers and the kill switch. Log in as `MM01` for the market maker's quote management.

5.4 When a display looks empty

- **TapeDeck shows symbols but no trades.** Nobody has traded yet. Quotes are not trades.
- **The Order Book Viewer shows an empty ladder.** The book has no orders. On `s3-basic-nomm` with the `default` profile that is the expected state until you trade.
- **A banner says the application is disconnected or reconnecting.** The exchange is stopped or restarting. Check `./edumatcher.sh status`.

5.5 Where to go next

Each application has its own chapter in the Participant Guide — TapeDeck, the Trading GUI and the Order Book Viewer — and the Operator's Guide covers the Log Operator Console, the Operator Dashboard and the Configuration GUI. Part II of this guide continues with Run a Trading Session.

CHAPTER 6

Run a Trading Session

Learning objectives

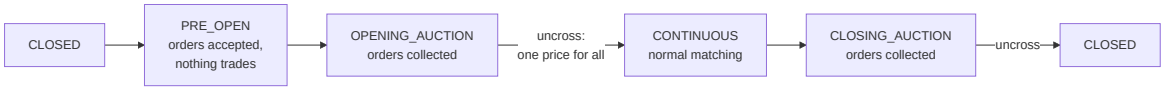
After this chapter you will be able to:

- name the phases of a trading day and say what each does to an order
- move the exchange through those phases yourself from the operator console
- see an opening auction collect orders and match them at a single price
- find the records the exchange keeps of everything that happened

Time: about 45 minutes. **You need:** the exchange from Install and Start and Your First Trade behind you.

6.1 A trading day has phases

So far the exchange has been open for continuous trading all the time. Real exchanges are not: they open and close on a timetable, and they use **auctions** to start and end the day. EduMatcher follows the same pattern.



Phase	What happens to orders
`CLOSED`	Everything is rejected: *Market is closed.*
`PRE_OPEN`	Orders are accepted and rest in the book, but **nothing trades**, even if prices cross.
`OPENING_AUCTION`	Still no trading. The engine works out the single price at which the most shares can change hands.
`CONTINUOUS`	The auction **uncrosses** at that price, then normal one-by-one matching runs all day.
`CLOSING_AUCTION`	Orders are collected again for a closing price.
back to `CLOSED`	The closing auction uncrosses; `DAY` orders expire.

Why the auctions? At the open, overnight news means nobody knows the right price. Letting

everyone's orders gather first and then trading all of them at one price is fairer than rewarding whoever happens to arrive first, and it produces an official opening and closing price.

6.2 Step 1 — Switch to a configuration with a timetable

The configuration `s3-nominal-nomm` has the same three symbols and four participants, an empty book, and a weekday timetable: pre-open 09:00, opening auction 09:25, continuous trading 09:30, closing auction 16:00 to 16:05. It also switches on circuit breakers, which you will not trip today.

Containers:

```
cd ~/.edumatcher
./edumatcher.sh config s3-nominal-nomm
./edumatcher.sh restart
```

(Keep `EM_PROFILE=default` from the previous chapter, so the market-maker bot stays off.)

Python package:

```
pm-opctl-cli stop
pm-setup --config s3-nominal-nomm --force
pm-opctl-cli start
```

The process set includes `pm-scheduler`, which moves the exchange through the phases at the times in the timetable — in the container's time zone, which is `Europe/Stockholm` unless you set `TZ` in `.env`. Outside those hours the exchange simply sits in `CLOSED`. In this chapter you will drive the phases by hand, which works at any time of day.

Check the clock first

The timetable applies on weekdays. If you start between 09:00 and 16:05 Stockholm time on a weekday, the scheduler has already moved the exchange into the pre-open, auction or continuous phase, and the next step (a rejected order in `CLOSED`) will not happen. Run `SESSION_STATUS` in Step 2 to see where you are. You can still follow the chapter: drive the phases by hand from wherever the exchange is, but remember that the scheduler will move it on again at the next time in the timetable. On a weekend, or in the evening, the exchange stays in `CLOSED` and everything reads exactly as written. In a classroom, run this chapter outside trading hours.

6.3 Step 2 — Open the operator console

In a shell inside the exchange, connect as the operator `OPS01`:

```
pm-admin --id OPS01
```

`pm-admin` is the **operator console**. It can see and do things a trader cannot: show the whole book, change the session phase, halt a symbol, disconnect a participant. Its prompt is `[OPS01|ADMIN]>`. Ask where the day is:

```
[OPS01|ADMIN]> SESSION_STATUS
  Session state     : CLOSED
  Auto-scheduling   : ON
[OPS01|ADMIN]> SCHEDULE
```

`SCHEDULE` prints the timetable. Two more commands give you the operator's view of the exchange:

```
[OPS01|ADMIN]> SYMBOLS
[OPS01|ADMIN]> GATEWAYS
```

`GATEWAYS` lists every participant with its role and whether it is connected right now.

Open two trader consoles as in the previous chapter (`pm-alf-console --id TRADER01` and `--id TRADER02`) and try to buy something from `TRADER01`:

```
[TRADER01]> NEW|SYM=AAPL|SIDE=BUY|TYPE=LIMIT|QTY=100|PRICE=150.00|TIF=DAY
[09:58:10.822] REJECTED  334087acbd9d91957322d02fc9db99a7 code=MARKET_CLOSED  Market is closed
```

That is the `CLOSED` phase doing its job.

6.4 Step 3 — Pre-open: orders wait

Move the exchange to pre-open. From `CLOSED` this is the only legal step:

```
[OPS01|ADMIN]> SESSION|STATE=PRE_OPEN
SESSION  CLOSED → PRE_OPEN
```

Now send a buy at 150.00 from `TRADER01` and a sell at 149.00 from `TRADER02`:

```
[TRADER01]> NEW|SYM=AAPL|SIDE=BUY|TYPE=LIMIT|QTY=100|PRICE=150.00|TIF=DAY
```

```
[TRADER02]> NEW|SYM=AAPL|SIDE=SELL|TYPE=LIMIT|QTY=100|PRICE=149.00|TIF=DAY
```

Both are acknowledged, and **neither trades** — even though the buyer would pay more than the seller asks. In continuous trading these two orders would have matched instantly. In pre-open they wait.

6.5 Step 4 — The opening auction and the uncross

Step through the auction into continuous trading:

```
[OPS01|ADMIN]> SESSION|STATE=OPENING_AUCTION
SESSION  PRE_OPEN → OPENING_AUCTION
[OPS01|ADMIN]> SESSION|STATE=CONTINUOUS
SESSION  OPENING_AUCTION → CONTINUOUS
```

At the moment continuous trading starts, the auction **uncrosses**. Both traders receive a fill:

```
[09:59:13.304] FILL      0aeb15eef2df8ba63f2eaf806e628600  qty=100 @149.0  remaining=0  [FILLED]
```

Every order matched in an auction trades at **one price**, the **equilibrium price**: the price at which the largest number of shares can change hands. If several prices allow the same volume, EduMatcher prefers the one that leaves the smaller imbalance (unmatched shares), and if that is also a tie it picks the lowest price. Here any price from 149.00 to 150.00 would let all 100 shares trade with nothing left over, so the lowest wins and both fill at 149.00. With more orders on each side the choice becomes a real calculation, and the Training Guide's Auctions chapter has you work one out by hand.

Phases only move forward

The engine accepts only the transitions in the order shown in the diagram. A jump such as `CLOSED` straight to `CONTINUOUS` is refused. If a `SESSION|STATE=` command seems to do nothing, run `SESSION_STATUS` to see where you really are.
Finish the day the same way — `SESSION|STATE=CLOSING_AUCTION`, then `SESSION|STATE=CLOSED` — and any `DAY` orders still resting expire.

Hand control versus the scheduler

`pm-scheduler` only acts at the times in the timetable. Between those times your manual changes stand; at the next scheduled time the scheduler moves the exchange on again. When you want the clock to run the day for you, just leave it alone during trading hours.

6.6 Step 5 — Look at the records

The exchange records everything while it runs. From a shell inside it:

```
pm-stats-cli trades           # every matched trade
pm-stats-cli daily            # open, high, low, close and volume per symbol and day
pm-clearing-cli pnl           # positions and profit or loss per participant
pm-audit-cli events --limit 20    # the last 20 events of the full audit trail
```

These read the databases written by `pm-stats`, `pm-clearing` and `pm-audit`. They are ordinary files in the data directory, so you can also open them with any SQLite tool.

And to watch the market live in the terminal rather than the browser:

```
pm-viewer --symbol AAPL      # one order book in detail
pm-board                     # several symbols at once
pm-ticker                    # a scrolling tape of trades
```

6.7 Checkpoint

- `SESSION_STATUS` shows the phase you set.
- Your pre-open orders traded only when continuous trading started, both at the same price.
- `pm-stats-cli trades` lists that trade.

6.8 Where to read more

If you want to know…	Read
Every phase, the scheduler, holidays and the trading date	Operator's Guide, Session Scheduling and Auctions
How the trading day looks from the trader's seat	Participant Guide, A Full Trading Day
Everything the operator console can do	Operator's Guide, The Admin Console and Exchange Commands
Halts, price collars and circuit breakers	Operator's Guide, Risk Controls
What each record contains	Operator's Guide, Statistics and Reporting and Audit Trail

CHAPTER 7

Your Own Configuration

Learning objectives

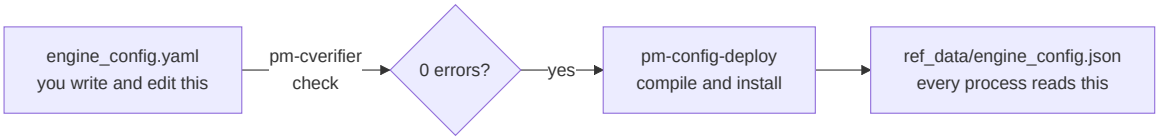
After this chapter you will be able to:

- explain the difference between the file you edit and the file the exchange runs
- generate a configuration with your own symbols and participants
- check it with the configuration verifier before you deploy it
- deploy it, on either installation route, and confirm it is running

Time: about 30 minutes.

7.1 The one idea to understand first

EduMatcher separates **the file you edit** from **the file it runs**:



You write YAML. `pm-config-deploy` checks it, compiles it and installs the compiled copy in the data directory, where every `pm-*` process reads it at startup. Editing the YAML alone changes nothing until you deploy it and restart. That is deliberate: in a system of many processes, the alternative is processes that disagree about the rules.

7.2 Step 1 — Generate a starting point

You rarely write a configuration from scratch. `pm-config-gen` writes a complete, valid one from a few options:

```
pm-config-gen \
    --symbols AAPL MSFT TSLA NOKIA \
    --participants TRADER01:TRADER TRADER02:TRADER OPS01:ADMIN MM01:MARKET_MAKER \
    --seed-mm-mid-range 20:200 \
    --seed-last-prices-from-mm \
    --api-gateway-instance desk:TRADER01,TRADER02,MM01,OPS01:8080 \
    --api-gateway-instance dashboards::8081 \
    --api-gateway-readonly-key \
    --market-data-gateway --market-data-enabled \
    --output my_exchange.yaml
```

What the options do:

Option	Effect
`--symbols`	The instruments to trade; each gets its own order book
`--participants ID:ROLE`	Who may connect, and as what: `TRADER`, `MARKET_MAKER` or `ADMIN`
`--seed-mm-mid-range 20:200`	Gives `MM01` a starting quote on every symbol around a random price between 20 and 200. A participant with the `MARKET_MAKER` role must have one, unless you ask for an empty book with `--no-mm-seed-quotes`
`--seed-last-prices-from-mm`	Uses the same prices as each symbol's reference price, which the price checks (collars) need
`--api-gateway-instance …`, `--api-gateway-readonly-key`	Creates API keys, so the Trading GUI and the market displays can log in
`--market-data-gateway --market-data-enabled`	Turns on the market-data feed that TapeDeck uses
`--output`	Where to write the YAML

Open `my_exchange.yaml` in an editor and read it: every section carries comments. `pm-config-gen --help` lists the many other options — schedules, risk limits, combos, an index — and the Configuration GUI at http://localhost:8092 does the same job with forms.

On the container route

The commands in this chapter run inside the exchange, so write the file there and copy it out, or run `pm-config-gen` on your own machine if you also have the Python package. The simplest container-only way is to start from the configuration that is already running:

```
./edumatcher.sh shell cat /data/ref_data/engine_config.yaml > my_exchange.yaml
```

and edit that file.

7.3 Step 2 — Check it

```
pm-cverifier my_exchange.yaml
```

`pm-cverifier` checks the YAML, every field, and the rules between fields — a market maker without a quote, a schedule with phases out of order, a symbol whose risk settings contradict each other — and explains each problem in plain words. A clean run ends with:

```
Verdict:  ✓ OK — no issues found
```

Warnings and advisories are fine to leave alone at this stage; **errors** must be fixed, because the engine will refuse to start. Running the verifier every time is much faster than finding a problem when the engine will not start.

7.4 Step 3 — Deploy and restart

Python package:

```
pm-config-deploy my_exchange.yaml    # checks, compiles and installs it
pm-opctl-cli stop
pm-opctl-cli start
```

Containers:

```
cd ~/.edumatcher
./edumatcher.sh config ./my_exchange.yaml
./edumatcher.sh shell pm-cverifier /config/engine_config.yaml
./edumatcher.sh restart
```

`./edumatcher.sh config` copies your file into `~/.edumatcher/config`, where the exchange container sees it as `/config/engine_config.yaml`; the restart deploys it.

7.5 Checkpoint

After the restart, ask the running exchange what it was configured with:

```
pm-config-show
```

The panels list your four symbols and four participants. TapeDeck shows the new symbols, and `NOKIA` has a market maker's quote on it.

7.6 Where to read more

Topic	Read
The whole configuration workflow, with complete examples	Operator's Guide, The Configuration Workflow
Every check the verifier makes	Operator's Guide, Configuration Verifier
A form-based editor instead of YAML	Operator's Guide, Configuration GUI
Ready-made configurations to copy	Operator's Guide, Example Engine Configs
Every field, its type and its default	Reference Manual, Configuration Schema
Why a market maker must quote, and how to opt out	Participant Guide, Market-Maker Quotes

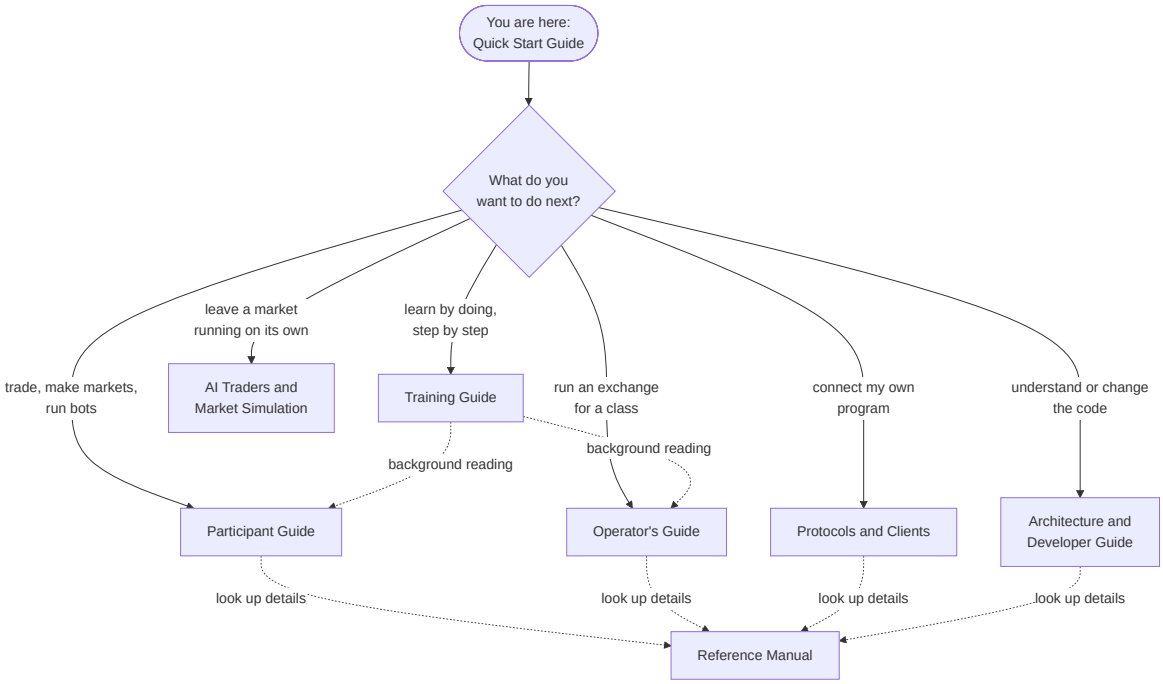
CHAPTER 8

Choose Your Next Book

Learning objectives

After this chapter you will know which book to open next for what you want to do, and in what order to read it.
You have run an exchange, traded on it, operated a trading day and deployed a configuration of your own. From here there is no single order: pick the path that matches what you want to do. A rule of thumb for the whole library: **read a chapter when you have a question it answers**, not before. Only the Training Guide is meant to be worked through from start to finish.

8.1 The library at a glance



Book	Open it when you want to…
Training Guide	learn every feature through exercises with checkpoints

Book	Open it when you want to…
Participant Guide	trade: order types, auctions, combo orders, market making, bots, P&L, the trading screens
Operator's Guide	install, configure, run, supervise and recover an exchange
Protocols and Clients	write a program that trades on, or listens to, the exchange
Architecture and Developer Guide	understand how EduMatcher is built, or change it
Reference Manual	look up one command, option, configuration field or term
AI Traders and Market Simulation	run a market that trades on its own — AI traders, hidden values, news — for days or weeks
How an Exchange Works	understand exchanges in general, without any software

8.2 Reading paths by role

8.2.1 Student learning to trade

1. How an Exchange Works, if markets are new to you.
2. Training Guide, chapters 00–08: placing, amending and cancelling orders, order types, time in force and auctions.
3. Participant Guide, as background whenever an exercise raises a question — start with The Order Book and Order Types.

8.2.2 Instructor running a class

1. Operator's Guide, Parts I–III: installation, configuration and Running the Exchange.
2. Operator's Guide, Session Scheduling and Auctions and Risk Controls.
3. Training Guide, chapters 00–19, to know exactly what your students will do.
4. AI Traders and Market Simulation, to make a classroom market look alive — or leave one running all term.

8.2.3 Market maker

1. Participant Guide, Part III: Market-Maker Quotes, Market Making and The Market-Maker Bot.
2. Training Guide, chapter 09 (Market Making) and chapter 22 (bot tuning).

8.2.4 Operator or supervisor

1. Operator's Guide, Running the Exchange and The Admin Console and Exchange Commands.
2. Risk Controls, then Part VI: persistence, the audit trail and the log server.

8.2.5 Analyst or auditor

1. Participant Guide, Positions and P&L.
2. Operator's Guide, Statistics and Reporting, Audit Trail and Audit Replay.

8.2.6 Developer of a client, feed handler or dashboard

1. Protocols and Clients, Protocols Overview, which tells you which protocol fits your purpose. For a dashboard the REST and WebSocket API is usually the right answer.
2. The specification of that protocol in Part II, and the session behavior chapter in Part III.
3. Training Guide, chapters 23–28, for exercises against a live gateway.

8.2.7 Contributor to EduMatcher itself

1. Architecture and Developer Guide, Architecture and the Guided Tour.
2. Development Practice and The Development Loop.

8.3 Depth by interest

Automated traders. Bots are the easiest way to get a book that looks alive without typing orders: see the Participant Guide's AI Traders and The Market-Maker Bot.

The internals. The Architecture and Developer Guide's Order Book Deep Dive and Order Flow Through the Engine follow one order through the code.

Everything at once. The Training Guide covers every feature, in order, 30 chapters.

CHAPTER 9

When Something Does Not Work

Try these in roughly this order. Most first-day problems are one of the first three.

9.1 Quick checks

Question	Containers	Python package
Is everything running?	`./edumatcher.sh status`	`pm-opctl-cli list`
What did the exchange say when it started?	`./edumatcher.sh logs`	the log files under `<data directory>/emo/` (the directory itself is in `pm-opctl-cli show`)
Which configuration is actually deployed?	`./edumatcher.sh shell pm-config-show`	`pm-config-show`
What is each process doing?	Log Operator Console at http://localhost:8091	`pm-log-cli`
Am I looking at the exchange I think I am?	`./edumatcher.sh mounts`	`pm-setup --show`
Which ports are published?	`./edumatcher.sh urls`	—

9.2 Common first-day problems

The installer or `./edumatcher.sh start` fails, or a container is missing. Run `./edumatcher.sh status` to see which container is down and `./edumatcher.sh logs` (press `Ctrl-C` to stop watching) for the reason. The usual causes are:

- *Neither Podman nor Docker is installed*, or Docker has no Compose support. Check with `podman --version` or `docker compose version`. On macOS the Docker Desktop application, or `podman machine start`, must be running.
- *The images could not be downloaded.* Check your internet connection and run `./edumatcher.sh pull`, then `./edumatcher.sh start` again.
- *Permission denied on the Docker socket* (Linux). Your user must be allowed to use Docker (see Docker's "post-installation steps for Linux"), or use Podman, which needs no special rights.
- *A port is already in use.* The exchange publishes 8080, 8081 and 8090–8095 on your computer. Stop whatever else uses them, or change a browser application's port in `.env` (for example `TERMINAL_GUI_PORT=8190`) and run `./edumatcher.sh restart`; `./edumatcher.sh urls` prints the

addresses in use. Another running EduMatcher installation counts too (see *Two installations at once* below).

A browser page does not open, or stays empty. Open the address on the same computer that runs the exchange: it listens on `localhost` only. If the page opens but shows a disconnected banner, check `./edumatcher.sh status`. An empty page can also be correct: see *When a display looks empty* in The Browser Applications.

The Trading GUI does not accept my API key. Each configuration has its own keys, so after you switch configuration the old ones stop working. Look them up again with `./edumatcher.sh shell pm-config-show -a`. Paste the whole key, with no spaces, and use the key of the participant you mean.

The console says *Gateway already connected.* A participant ID can be connected once at a time. Another console, or a Trading GUI login, is using it. Close that one, or use another participant such as `TRADER02`.

My orders disappeared. When a trader disconnects (typing `EXIT`, or closing the console window) the exchange cancels that trader's resting orders in the bundled configurations. Orders with `TIF=GTC` are cancelled too; they only survive a stop and start of the *exchange*, not a trader leaving.

I want to start over. Reset the saved state and keep the configuration: stop the processes, clear them, restart.

```
./edumatcher.sh shell pm-opctl-cli stop
./edumatcher.sh shell pm-opctl-cli clear --state --yes
./edumatcher.sh restart
```

On the Python route use `pm-opctl-cli stop`, `pm-opctl-cli clear --state --yes` and `pm-opctl-cli start`. `clear --all` also removes logs and audit files.

My order does not trade. Nobody is waiting on the other side at a price that crosses yours. That is the market working correctly. Look at the book (`pm-viewer --symbol AAPL`, `BOOK|SYM=AAPL` in `pm-admin`, or the Order Book Viewer) to see the best bid and ask.

My order traded immediately, before the other trader did anything. The book already had liquidity: a market maker's seeded quote, or the market-maker bot of the container's `mm-demo` profile. Use an `-nomm` configuration and the `default` profile, as in Your First Trade, to start from an empty book.

Every order is rejected with *Market is closed.* The configuration has a timetable and the exchange is outside trading hours, or was left in `CLOSED`. Move it on by hand from `pm-admin` (`SESSION|STATE=PRE_OPEN`, then `SESSION|STATE=CONTINUOUS`), as in Run a Trading Session, or use a `-basic` configuration, which has no timetable.

The console will not connect, or says the participant is unknown. Participant IDs are case-sensitive and must exist in the deployed configuration; `pm-config-show` lists them. On the container route, run the console inside the exchange (`./edumatcher.sh shell`), not on your own machine.

`pm-*: command not found.` On the container route the commands only exist inside the exchange: open `./edumatcher.sh shell` first. On the Python route, check that `pipx` put its directory on your `PATH` (`pipx ensurepath`).

A recorder's numbers are missing trades. `pm-stats`, `pm-clearing` and `pm-audit` only record what happens while they are running. Start them before the engine, which `pm-opctl-cli` and the container do for you.

I changed the configuration but nothing changed. A configuration must be deployed and the processes restarted. On the container route, `./edumatcher.sh config …` only takes effect after

`./edumatcher.sh restart`.

Two installations at once. A curl installation and a stack built from the source checkout use the same container names and ports, so only one can run. Both refuse to start on top of the other; `./edumatcher.sh mounts` shows which one owns the ports.

9.3 Still stuck?

The FAQ answers the next layer of questions, and the Operator's Guide has a full troubleshooting section in Running the Exchange. If something in this guide does not work as written, that is a documentation bug — please report it on GitHub.

CHAPTER 10

FAQ & Common Mistakes

Common questions and gotchas when starting with EduMatcher. For the first checks to make when something does not work at all, start with When Something Does Not Work. Questions that come up later — order types, saved orders, seed quotes, the scheduler, configuration files — are listed at the end under More questions.

10.1 Architecture & Setup

10.1.1 The engine started but gateways can't connect. What's wrong?

Check that: 1. The engine is running *before* the gateways try to connect. 2. ZMQ ports 5555 and 5556 are not blocked by a firewall or in use by another process: `lsof -i :5555 -i :5556` 3. The participant ID exists in the deployed configuration (`pm-config-show` lists them). 4. On the container route, the console runs *inside* the exchange (`./edumatcher.sh shell`). The message bus is not published to your own machine unless you set `EM_ZMQ=1`.

If the engine is down or unreachable, `pm-alf-console` exits after about three seconds with `Gateway authentication timed out`.

10.1.2 The book already has liquidity before I submit any orders. Why?

If your `engine_config.yaml` has a `market_maker_quotes` block for the symbol, the engine injects a two-sided seed quote on behalf of the configured market-maker gateway at startup. This is intentional — it gives the book immediate liquidity so participants always have something to trade against.

A side effect is that your **first aggressive order may fill immediately** against the seed quote, before a second participant has typed anything.

To run with a deliberately empty book (e.g. for a demo of price discovery from scratch), either: - deploy one of the `-nomm` example configurations, such as `s3-basic-nomm` (`pm-setup --config s3-basic-nomm --force`, or `./edumatcher.sh config s3-basic-nomm` on the container route), or - generate your own with `pm-config-gen --no-mm-seed-quotes`.

On the container route, also note that the default `mm-demo` process profile runs a market-maker bot that quotes every symbol; set `EM_PROFILE=default` in `.env` to run without it.

10.1.3 Why is the engine not matching orders? It starts in CLOSED state

With `sessions_enabled: true` (the default), the engine starts in `CLOSED` state and waits for `pm-scheduler` to drive it through the session phases before it accepts orders.

Three ways to get to a matching state quickly:

1. **Start `pm-scheduler`** alongside the engine — it drives the transition from CLOSED → PRE_OPEN → OPENING_AUCTION → CONTINUOUS automatically at the configured wall-clock times, or immediately with `pm-scheduler --now`.
2. **Move the phase by hand** from the operator console: `pm-admin --id OPS01`, then `SESSION|STATE=PRE_OPEN` and `SESSION|STATE=CONTINUOUS`. See the Quick Start's Run a Trading Session.
3. **Disable sessions** for testing or demos:

   ```
   # engine_config.yaml
   sessions_enabled: false
   ```

   With `sessions_enabled: false` the engine enters CONTINUOUS state immediately on startup and `pm-scheduler` has no effect.

10.1.4 Why is my viewer empty even though it started correctly?

An empty viewer usually means the symbol has no resting orders yet. `pm-viewer` subscribes to `book.<SYMBOL>` and then requests an initial snapshot, but if the book is empty the snapshot is also empty.

Fix: Submit a resting order, enable market-maker quote seeds for that symbol, or verify that you are watching a configured symbol.

10.1.5 Do I really need a terminal window for every process?

No. Each process is an independent program communicating over ZeroMQ, and each blocks on its own event loop, so started by hand each needs its own terminal. But you rarely need to start them by hand: `pm-opctl-cli start` starts a whole named profile of processes in the background, in the right order, and `pm-opctl-cli list` shows their health. The container does exactly this for you. Open terminals only for what you interact with: trader consoles, the operator console and viewers. `tmux` or `screen` help when you do want many side by side.

The source checkout also has `tools/launch_all.sh`, which opens every process in its own macOS Terminal window. It uses `osascript`, so it only works on macOS.

10.1.6 How do I reset all persisted state to start fresh?

Stop the exchange, then let `pm-opctl-cli` clear the data directory:

```
pm-opctl-cli stop
pm-opctl-cli clear --state    # order books, GTC orders, stats, index and clearing state
pm-opctl-cli clear --all      # ...and the logs and audit files as well
```

The deployed configuration under `ref_data/` is never touched. On the container route, run the same commands with `./edumatcher.sh shell` while the exchange is running, then `./edumatcher.sh restart`.

After a reset the engine re-seeds market-maker quotes as if it were day 1 (because `book_stats.json` no longer exists).

10.2 Orders, Positions & Sessions

10.2.1 Why is my LIMIT order filling at a different price than I specified?

Your limit price is the **worst** price you'll accept — not necessarily the price you'll get. If better prices are available, you fill at those better prices.

Example: You submit `BUY LIMIT QTY=100 PRICE=151.00` but the book has asks at 149.90 and 150.20. You fill at 149.90 and 150.20 (both better than your limit of 151.00), not at 151.00.

This is called **price improvement** — you get a better deal than the worst price you were willing to accept.

10.2.2 I submitted a MARKET order but it only partially filled. Where did the rest go?

A MARKET order sweeps the book until it runs out of liquidity. If the book does not have enough resting orders to fill the entire quantity, the unfilled remainder is **discarded** (not rested). You will receive one or more partial fill events, then a cancellation for the remaining quantity.

Fix: If you need guaranteed full execution at whatever price, use FOK and ensure liquidity is available. If you're comfortable with partial fills, MARKET is correct — just understand the remainder is discarded, not queued.

10.2.3 Why is my unrealized P&L different from my realized P&L?

They measure different things:

- **Unrealized P&L**: paper gain/loss on shares you still hold, calculated as `(current_price - avg_cost) × position`. This changes every time a trade prints and updates the last trade price.
- **Realized P&L**: profit/loss locked in when you *closed* part of your position (sold shares you were long, or bought back a short). This never changes after the trade.

Until you close your position, your gains are "on paper" — the market can take them back. Once realized, they are yours regardless of subsequent price moves.

10.2.4 Why do I have a negative position?

A negative position means you are **short** — you have sold shares you did not own. This can happen if you submit a SELL order without first having bought any shares. In a real brokerage this requires a margin account and stock borrowing; EduMatcher does not enforce these constraints, so short positions are allowed freely.

10.2.5 What happens to my open orders when the market closes?

It depends on their time-in-force:

TIF	At market close
`DAY`	Expired and removed. You receive `order.expired` events.
`GTC`	Saved to `data/gtc_orders.json` and reloaded on next engine start.
`ATO`	Expired when OPENING_AUCTION phase ends.
`ATC`	Expired when CLOSING_AUCTION phase ends.

10.3 Installation

10.3.1 Do I need Poetry or pipx?

It depends on your role:

Role	Tool	Install command
Student / instructor (just want to run the exchange)	pipx	`pipx install edumatcher`
Developer (modifying the engine, running tests)	Poetry	`poetry install --with dev`

End users should not need a source checkout at all. After `pipx install edumatcher` all `pm-*` commands are on your PATH and ready to use.

10.3.2 I installed with pipx but `pm-engine` is not found. What do I do?

pipx installs scripts to `~/.local/bin`, which may not be on your PATH yet.

```
pipx ensurepath        # adds ~/.local/bin to your shell profile
exec $SHELL            # reload your current shell
which pm-engine        # should now print a path
```

If you just installed pipx itself for the first time, open a **new terminal window** after running `pipx ensurepath` — the PATH change only takes effect in new shells.

10.3.3 What does `pm-setup` do and do I have to run it?

`pm-setup` is a one-time bootstrap command for **installed** (pipx) users:

1. Creates the data directory (`~/.local/share/edumatcher` by default).
2. Compiles a bundled example configuration — `s10-basic` unless you choose another with `--config` — and deploys it to `<data directory>/ref_data/engine_config.json`, keeping the YAML source beside it.
3. Prints the `export EDUMATCHER_DATA_DIR=...` line to add to `~/.zshrc` or `~/.bashrc`.

Without a deployed configuration no process will start, so you need either `pm-setup` or your own `pm-config-deploy`. An already-deployed configuration is kept unless you pass `--force`. The container installation runs the equivalent for you.

10.3.4 How do I upgrade EduMatcher to a newer version?

```
pipx upgrade edumatcher            # Python package, from PyPI
./edumatcher.sh update             # container installation, latest release
```

Your configuration and data files are not touched by the upgrade. If the new version changes the configuration format, redeploy your configuration with `pm-config-deploy` (or, for a bundled example, `pm-setup --config <name> --force`). Do not run `pm-setup --force` otherwise: it replaces the deployed configuration with a bundled example.

10.4 More questions

These questions are less common on a first day, so each lives in the book that covers its topic. Nothing has been dropped.

Question	Where
Why was my IOC order rejected?	Participant Guide
Why didn't my STOP order trigger?	Participant Guide
Why did my STOP_LIMIT order trigger but not fill?	Participant Guide
My FOK order was rejected even though there was liquidity. Why?	Participant Guide
Why does my average cost change when I buy more shares?	Participant Guide
My GTC orders didn't reload after restart. Why?	Participant Guide
What is SMP and why did my order get cancelled?	Participant Guide
Why does the engine use integer ticks instead of float prices?	Participant Guide
Why did the seed MM quote disappear after the second day?	Operator's Guide
Why did `pm-scheduler` exit immediately?	Operator's Guide
How do I see what messages are flowing between processes?	Operator's Guide
Why is my order rejected with "Gateway not authenticated"?	Operator's Guide
The engine says it cannot find its configuration. Where should the file be?	Operator's Guide
Where are my data files stored?	Operator's Guide

CHAPTER 11

Quick Glossary

The words this guide uses, in one page. The Reference Manual's Glossary has the full vocabulary.

Term	Meaning
Ask	The lowest price at which someone is currently willing to sell. Also called the *offer*.
Auction	A phase in which orders are collected and then all matched at once, at one price (the *equilibrium price*). Used to open and close the day.
Bid	The highest price at which someone is currently willing to buy.
Circuit breaker	A risk control that halts trading in a symbol after a large, fast price move.
Configuration	The YAML file that describes the whole exchange: symbols, participants, timetable and rules. It is *deployed* before the exchange uses it.
Drop copy	An independent copy of every fill, sent to risk and compliance systems.
Engine	`pm-engine`, the one process that owns the order books and matches orders.
Fill	An execution: part or all of an order traded.
Gateway	A process through which participants connect, or the connection itself. The participant ID is also called the *gateway ID*.
Limit order	An order with a price limit: buy at that price or lower, sell at that price or higher.
Maker / taker	The maker's order was resting in the book; the taker's order arrived and traded against it.
Market maker	A participant whose job is to quote both a bid and an ask, so others can always trade.

Term	Meaning
Market order	An order without a price: trade now at the best prices available.
Order book	All resting buy and sell orders for one symbol, best prices first.
Participant	A trader, market maker or operator configured to connect, with an ID such as `TRADER01` and a role.
P&L	Profit and loss. *Realized* P&L is locked in by closing a position; *unrealized* P&L is what an open position would make at the last price.
Position	How many shares a participant holds: positive is *long*, negative is *short*.
Price-time priority	The matching rule: better prices first; at the same price, earlier orders first.
Resting	An order waiting in the book for someone to trade with it.
Session phase	Where the trading day is: `PRE_OPEN`, `OPENING_AUCTION`, `CONTINUOUS`, `CLOSING_AUCTION`, `CLOSED`.
Spread	The gap between the best ask and the best bid.
Symbol	A tradeable instrument, such as `AAPL`, with its own order book.
Tick	The smallest allowed price step for a symbol, for example 0.01.
Time in force (TIF)	How long an order stays active: `DAY`, `GTC` (good till cancelled), or only in an auction (`ATO`, `ATC`). `IOC` and `FOK` are order *types*, not TIF values.
ALF, BALF, CALF, RALF, LALF	EduMatcher's protocols: text order entry, binary order entry, market data, post-trade data and logging.